The Hardness of Subgraph Isomorphism

Marek Cygan[*] Jakub Pachocki[†] Arkadiusz Socała[‡]

April 14, 2015

Abstract

Subgraph Isomorphism is a very basic graph problem, where given two graphs G and H one is to check whether G is a subgraph of H. Despite its simple definition, the Subgraph Isomorphism problem turns out to be very broad, as it generalizes problems such as Clique, r-Coloring, Hamiltonicity, Set Packing and Bandwidth. However, for all of the mentioned problems $2^{\mathcal{O}(n)}$ time algorithms exist, so a natural and frequently asked question in the past was whether there exists a $2^{\mathcal{O}(n)}$ time algorithm for Subgraph Isomorphism. In the monograph of Fomin and Kratsch [Springer'10] this question is highlighted as an open problem, among few others.

Our main result is a reduction from 3-SAT, producing a subexponential number of sublinear instances of the Subgraph Isomorphism problem. In particular, our reduction implies a $2^{\Omega(n\sqrt{\log n})}$ lower bound for Subgraph Isomorphism under the Exponential Time Hypothesis. This shows that there exist classes of graphs that are strictly harder to embed than cliques or Hamiltonian cycles.

The core of our reduction consists of two steps. First, we preprocess and pack variables and clauses of a 3-SAT formula into groups of logarithmic size. However, the grouping is not arbitrary, since as a result we obtain only a limited interaction between the groups. In the second step, we overcome the technical hardness of encoding evaluations as permutations by a simple, yet fruitful scheme of guessing the sizes of preimages of an arbitrary mapping, reducing the case of arbitrary mapping to bijections. In fact, when applying this step to a recent independent result of Fomin et al.[CoRR abs/1502.05447 (2015)], who showed hardness of Graph Homomorphism, we can transfer their hardness result to Subgraph Isomorphism, implying a nearly tight lower bound of $2^{\Omega(n \log n / \log\log n)}$.

[*]Institute of Informatics, University of Warsaw, cygan@mimuw.edu.pl
[†]Carnegie Mellon University, pachocki@cs.cmu.edu
[‡]Institute of Informatics, University of Warsaw, a.socala@mimuw.edu.pl

1 Introduction

Perhaps the most basic relation between graphs is that of being a subgraph. We say that G is a subgraph of H if one can remove some edges and vertices of H, so that what remains is isomorphic to G. Formally, the question of one graph being a subgraph of another is the base of the Subgraph Isomorphism problem.

> Subgraph Isomorphism
> **Input:** undirected graphs G, H.
> **Question:** is G a subgraph of H, i.e., does there exist an injective function $g : V(G) \to V(H)$, such that for each edge $uv \in E(G)$ we have $g(u)g(v) \in E(H)$.

Subgraph Isomorphism is an important and very general question, having the form of a pattern matching – we will call G the *pattern graph* and H the *host graph*. Observe that several flagship graph problems can be viewed as instances of Subgraph Isomorphism:

- Hamiltonicity(G): is C_n (a cycle with n vertices) a subgraph of G?
- Clique(G,k): is K_k a subgraph of G?
- 3-Coloring(G) : is G a subgraph of $K_{n,n,n}$, a tripartite graph with n vertices in each of its three independent sets?
- VertexCover(G,k) : is G a subgraph of H, H being a full join between a clique of size k and an independent set of size $n - k$?

One can continue showing the richness of Subgraph Isomorphism by simple linear reductions from Bandwidth, Set Packing and several other problems.

All of the mentioned problems are NP-complete, and the best known algorithms for all the listed special cases work in exponential time. In fact, all those problems are well-studied from the exact exponential algorithms perspective [3, 4, 5, 6, 7], where the goal is to obtain an algorithm of running time $\mathcal{O}(c^n)$ for smallest possible value of c. Furthermore, the Subgraph Isomorphism problem was very extensively studied from the viewpoint of fixed parameter tractability, see [16] for a discussion of 19 different possible parametrizations. All the mentioned special cases of Subgraph Isomorphism admit $\mathcal{O}(c^n)$ time algorithms, by using either branching, inclusion-exclusion principle or dynamic programming. On the other hand, a simple exhaustive search for the Subgraph Isomorphism problem – numerating all possible mappings from the pattern graph to the host graph – runs in $2^{\mathcal{O}(n \log n)}$ time, where n is the total number of vertices of the host graph and pattern graph.

Therefore, a natural question is whether Subgraph Isomorphism admits an $\mathcal{O}(c^n)$ time algorithm. This was repeatedly posed as an open problem [1, 2, 8, 9, 11]. In particular, Fomin and Kratsch in their monograph [10] put the existence of $\mathcal{O}(c^n)$ time algorithm for Subgraph Isomorphism among the few questions in the open problems section.

Our results and techniques Our main result is a reduction which transforms a 3-SAT formula into a subexponential number of sublinear instances of the Subgraph Isomorphism problem. This implies that a $\mathcal{O}(c^n)$ time algorithm for Subgraph Isomorphism would imply a subexponential algorithm for 3-SAT, thus refuting the Exponential Time Hypothesis of Impagliazzo, Paturi and Zane [12, 13]. The Exponential Time Hypothesis is an established assumption; several interesting lower bounds have been found under this conjecture (see [14] for a survey).

Theorem 1.1 *There is no algorithm which solves* Subgraph Isomorphism *in* $2^{o\left(n\sqrt{\log n}\right)}$ *time, unless the Exponential Time Hypothesis fails.*

Our reduction can be broken into three steps:

- First, in Section 4, we preprocess the given 3-SAT formula and pack its variables and clauses into groups of logarithmic size. Importantly, we ensure that there is only a limited interaction between the groups by marking variables with colors – applying further steps of the reduction for an arbitrary grouping would not yield a superexponential lower bound for Subgraph Isomorphism.
- Next, in Section 5, we use the packing to create $2^{\mathcal{O}(n/\log n)}$ smaller instances of a variant of the Subgraph Isomorphism problem, where additionally vertices and edges have colors which have to be preserved by the mapping. This proves that the color variant of Subgraph Isomorphism admits a tight lower bound of $2^{\Omega(n \log n)}$ under the Exponential Time Hypothesis. In this step, we use a simple technique of guessing preimage sizes, which allows us to circumvent the usual technical difficulties of encoding valuations by permutations.
- Finally, in Section 6 we reduce the color version of Subgraph Isomorphism to the original variant, incurring an $\mathcal{O}(\sqrt{\log n})$ increase in the instance size.

We would like to note that very recently and independently, Fomin et al. [9], in an unpublished work, proved that under the Exponential Time Hypothesis there is no $2^{o(n \log h/ \log \log h)}$ time algorithm for a related problem called Graph Homomorphism. Graph Homomorphism has a similar definition to Subgraph Isomorphism, except that the mapping is not constrained to be injective (i.e., in a homomorphism many vertices of the pattern graph may be mapped to the same vertex of the host graph). One could think that Graph Homomorphism is a harder problem than Subgraph Isomorphism, as for example in [2] Amini et al. have shown that counting subgraphs can be reduced to counting homomorphisms. In fact, Fomin et al. [9] in their work about Graph Homomorphism mention the question about Subgraph Isomorphism as an open problem.

Theorem 1.2 *[9] There is no algorithm which solves* Graph Homomorphism *in* $2^{o(n \log h/ \log \log h)}$ *time, where* $h = \mathcal{O}(poly(n))$ *is the size of the host graph and* n *is the size of the pattern graph, unless the Exponential Time Hypothesis fails.*

In Section 7 we prove that by applying our simple scheme of guessing preimage sizes, one can transform an instance of Graph Homomorphism into an exponential number of instances of Subgraph Isomorphism.

Theorem 1.3 *Given an instance* (G, H) *of* Graph Homomorphism *one can in* $\mathcal{O}(2^n poly(n))$ *time create* 2^n *instances of* Subgraph Isomorphism *with* n *vertices, where* $n = |V(G)| + |V(H)|$*, such that* (G, H) *is a yes-instance iff at least one of the created instances of* Subgraph Isomorphism *is yes-instance.*

Note that Theorem 1.3, when combined with the lower bound of Fomin et al. quoted in Theorem 1.2, implies a stronger lower bound for Subgraph Isomorphism.

Corollary 1.4 *There is no algorithm which solves* Subgraph Isomorphism *in* $2^{o(n \log n/ \log \log n)}$ *time, unless the Exponential Time Hypothesis fails.*

2 Preliminaries

Notation We use the convention $[k] = \{0, \ldots, k-1\}$. All the graphs used in this article are undirected, however in edge colored graphs there might be several parallel edges between the same pair of vertices. We use standard graph notation – for an undirected graph G, by $V(G)$ we denote the set of vertices of G, whereas by $E(G)$ we denote the set of edges of G.

For a CNF-SAT formula φ let $\mathrm{Var}(\varphi)$ be the set of variables of φ, whereas $\mathrm{Clauses}(\varphi)$ is the set of clauses of φ.

By saying that two instances I, I' of some decision problems P and Q, respectively, are equivalent, we mean that I is a yes-instance of the problem P iff I' is a yes instance of the problem Q. In particular two formulas are equivalent iff they are either none of both of them are satisfiable.

To simplify the reduction we use the standard method of transforming a 3-SAT formula into an equivalent formula with exactly three different variables in each clause and each variable occurring in at most 4 clauses.

Lemma 2.1 *[18] Given a* 3-SAT *formula φ with m clauses one can transform it in polynomial time into a formula φ' with $\mathcal{O}(m)$ variables and $\mathcal{O}(m)$ clauses, such that φ' is satisfiable iff φ' is satisfiable, and moreover each clause of φ' contains exactly three variables and each variable occurs in at most 4 clauses of φ'.*

Exponential Time Hypothesis The Exponential Time Hypothesis, introduced by Impagliazzo, Paturi and Zane [12, 13], states that it is impossible to solve 3-SAT in time subexponential in the number of variables. Note that the $\mathcal{O}^\star()$ notation suppresses polynomial factors.

Conjecture 2.2 (Exponential Time Hypothesis [13]) *There exists a constant $c > 0$, such that there is no algorithm solving* 3-SAT *in time $\mathcal{O}^\star(2^{cn})$.*

One of the reasons why the Exponential Time Hypothesis became a robust tool for proving lower bounds is the Sparsification Lemma, which allows to reduce the number of clauses in a formula to be linear in the number of variables.

Lemma 2.3 (Sparsification Lemma [12]) *For each $\varepsilon > 0$ there exist a constants c_ε, such that any* 3-SAT *formula φ with n variables can be expressed as $\varphi = \vee_{i=1}^{t}\psi_i$, where $t \leq 2^{\varepsilon n}$ and each ψ_i is a* 3-SAT *formula with the same variable set as φ, but contains at most $c_\varepsilon n$ clauses. Moreover, this disjunction can be computed in time $\mathcal{O}^\star(2^{\varepsilon n})$.*

3 Overview

We define the *size* of a Subgraph Isomorphism instance to be the total number of vertices in the pattern and host graphs.

Definition 3.1 *We define the (c,t)-*Subgraph Isomorphism *problem as a generalization of* Subgraph Isomorphism *where every vertex of the pattern and host graphs is colored in one of c colors, and every edge is colored in one of t colors, and the mapping is restricted to preserving vertex and edge colors.*

In particular, Subgraph Isomorphism is the same as $(1,1)$-Subgraph Isomorphism. The pipeline of our lower bound consists of two steps. First, in Lemma 3.2, given a 3-SAT formula with n variables we construct a set of $2^{\mathcal{O}(n/\log n)}$ instances of $(\mathcal{O}(1), \mathcal{O}(\log n))$-Subgraph Isomorphism of $\mathcal{O}(n/\log n)$ size each. Note that the number of vertex colors is constant, whereas the number of edge colors is logarithmic. In the second step (Lemma 3.3) we reduce to the original variant of Subgraph Isomorphism, with an additional increase in the instance size by a factor of $\mathcal{O}(\sqrt{\log n})$, leading to a final size of $\mathcal{O}(n/\sqrt{\log n})$, which is sublinear.

Lemma 3.2 *Given a* 3-SAT *formula φ with n variables, where each variable occurs in at most* 4 *clauses and each clause involves exactly three variables, one can in $2^{\mathcal{O}(n/\log n)}$ time create a set $\mathcal{S}$ of $2^{\mathcal{O}(n/\log n)}$ instances of $(\mathcal{O}(1), \mathcal{O}(\log n))$-*SUBGRAPH ISOMORPHISM *of size $\mathcal{O}(n/\log n)$, such that φ is satisfiable iff any instance in $\mathcal{S}$ is satisfiable, and the host graph and the pattern graph have the same number of vertices for every instance in $\mathcal{S}$.*

Lemma 3.3 *An instance of (c, t)-*SUBGRAPH ISOMORPHISM*, where the host graph and the pattern graph have the same number of vertices, can be reduced to an equivalent instance of* SUBGRAPH ISOMORPHISM *with $\mathcal{O}(c\sqrt{t})$ times more vertices.*

Having the two lemmas above, which we prove in the remainder of this paper, we can prove Theorem 1.1.

Proof of Theorem 1.1: Assume that a $2^{o(n\sqrt{\log n})}$ time algorithm exists for the SUBGRAPH ISOMORPHISM problem, where $n = |V(G)| + |V(H)|$. For a given $\varepsilon > 0$, we show an algorithm solving a given 3-SAT formula φ with n variables in time $\mathcal{O}^\star(2^{3\varepsilon n})$, leading to a contradiction with the Exponential Time Hypothesis.

First, we sparsify the formula using Lemma 2.3 to obtain $\mathcal{O}^\star(2^{\varepsilon n})$ formulas ψ_i, each with n variables and $\mathcal{O}(n)$ clauses (where the hidden constant depends on ε). Consider each ψ_i independently. For a fixed ψ_i, we use Lemma 2.1 to obtain an equivalent formula ψ_i' with $\mathcal{O}(n)$ variables and clauses, with the additional property that each clause involves exactly three variables and each variable appears in at most 4 clauses. Consequently, the prerequisites of Lemma 3.2 are satisfied, and in $2^{\mathcal{O}(n/\log n)}$ time we can obtain a corresponding set $\mathcal{S}$ of $2^{\mathcal{O}(n/\log n)}$ instances of $(\mathcal{O}(1), \mathcal{O}(\log n))$-SUBGRAPH ISOMORPHISM of size $\mathcal{O}(n/\log n)$ each. Next, we apply Lemma 3.3 to transform each instance in $\mathcal{S}$ into an instance of SUBGRAPH ISOMORPHISM of size $\mathcal{O}(n/\sqrt{\log n})$, obtaining the set $\mathcal{S}'$. Finally, we apply the hypothetical $2^{o(n\sqrt{\log n})}$-time algorithm to the instances in $\mathcal{S}'$, leading to $2^{\mathcal{O}(n/\log n)}2^{o(n)} = 2^{o(n)}$ running time. Note that the total running time is $\mathcal{O}^\star(2^{\varepsilon n}) \cdot \mathcal{O}^\star(2^{\varepsilon n}) \cdot 2^{o(n)}$, which is not more than $\mathcal{O}^\star(2^{3\varepsilon n})$, as promised, hence the theorem follows. ■

We prove Lemma 3.2 in Section 5 and Lemma 3.3 in Section 6. However, before we describe the reduction, in Section 4 we present how to group clauses of a given 3-SAT formula in a way that allows a sublinear reduction to SUBGRAPH ISOMORPHISM.

4 Grouping clauses

As we already mentioned, when proving superexponential lower bounds based on the Exponential Time Hypothesis, we need to come up with a reduction producing an instance of SUBGRAPH ISOMORPHISM of sublinear size. In this section we show how to preprocess a given 3-SAT formula and partition its clauses into groups of logarithmic size. Our grouping is far from arbitrary, as we need to precisely control the interactions between clauses sharing the same variables.

Before we arrive at our main structural lemma, we need a simple step in which we assign colors to variables so that no clause contains two variables of the same color and moreover the counts of variables in each color are balanced. The proof of the following Lemma is contained in Appendix A.

Lemma 4.1 (♠) *Given an integer $k > 9$ and a 3–*SAT *formula φ with n variables, where each variable occurs in at most* 4 *clauses, we can color the variables of φ in polynomial time using at most k colors, so that no more than $\lceil n/(k-9) \rceil$ variables share the same color and no clause contains two variables of the same color.*

Having Lemma 4.1 we are ready to pack the clauses of a given 3-SAT formula into 2^k groups, which is the main structural insight in our reduction. It is important that no two clauses from the same group contain variables of the same color.

Lemma 4.2 *Given a 3*−SAT *formula φ with $n \geq 16$ variables, such that each clause involves exactly three variables and each variable occurs in at most 4 clauses, one can in polynomial time construct:*

- *a coloring $l : \mathrm{Var}(\varphi) \to [k]$ of the variables in φ into k colors, such that no two variables contained in a clause of φ share the same color, and*
- *a packing $h : \mathrm{Clauses}(\varphi) \to [2^k]$ of the clauses into 2^k groups indexed by $\{0, \ldots, 2^k - 1\}$, such that for any $i \in [2^k]$ no two clauses that are mapped to i contain variables of the same color,*

where $k := \lceil \log n - \log \log n \rceil + 9$.

Proof: Let l be the coloring guaranteed by Lemma 4.1. We slightly overload the notation and by $l(C)$ denote the set of colors of variables in $C \in \mathrm{Clauses}(\varphi)$.

We construct the packing h in a greedy manner. Consider all the clauses of $\mathrm{Clauses}(\varphi)$ one by one in an arbitrary order. When a clause $C \in \mathrm{Clauses}(\varphi)$ is processed, we find any group $i \in [2^k]$, such that the set of colors of variables appearing in clauses already assigned to i is disjoint from $l(C)$. If several such sets i exist, we pick an arbitrary one and assign $h(C) := i$.

It remains to prove that such an i always exists for the value of k as stated in the lemma. We prove this by contradiction: suppose that at some point, for some clause C, for every i one of the colors in $l(C)$ is already present in a clause already assigned to i. Let $m_{l(C)}$ be the number of clauses of φ containing at least one color from $l(C)$. As there are exactly 2^k groups, and we cannot assign C to any of them, it means that

$$m_{l(C)} \geq 2^k \geq 512n/\log n\,, \tag{4.1}$$

since each of the 2^k groups is blocked by a different clause containing at least one color from $l(C)$.

On the other hand we have only 3 colors in $l(C)$ and we know by Lemma 4.1, that no more than $\lceil n/(k-9) \rceil$ variables are assigned to any color, and by the upper bound on the frequency of each variable of φ we know that no variable occurs in more than 4 clauses. Consequently, the number of clauses having at least one common color with C is upper bounded by

$$\begin{aligned} m_{l(C)} &\leq 3 \cdot \lceil n/(k-9) \rceil \cdot 4 \leq 12 \cdot (n/(k-9) + 1) \\ &\leq 12 \cdot (n/(\log n - \log\log n) + 1) \leq 12 \cdot (2n/\log n + 1) \\ &\leq 12 \cdot (2n/\log n + 0.5n/\log n) \leq 30n/\log n\,, \end{aligned} \tag{4.2}$$

where in the last two inequalities we have used that $\log n - \log\log n \geq 0.5 \log n$ and $n/\log n \geq 2$ for $n \geq 16$. Note that (**??**) yields a contradiction with (**??**), and the lemma follows. ■

5 From 3-SAT to Subgraph Isomorphism with colors

The technical crux of our result is a method of encoding information in permutations – mappings from the pattern graph to the host graph. The intuition behind this technique is that the number of permutations of an n element set is $n! = 2^{\Theta(n \log n)}$ and therefore a single permutation carries $\Theta(n \log n)$ bits of information. This means that from the information-theoretic perspective if should be possible to encode an assignment of Boolean values to n variables using a permutation of $\mathcal{O}(n/\log n)$ elements.

Every element in a permutation is responsible for encoding some number of bits, forming what we call a *pack* of bits. We do not restrict ourselves to packs of constant size, but each pack we create is of size no greater than logarithmic. The position of an element in a permutation should uniquely determine the values of all the bits from its pack. The problem is, however, that it in a permutation no two elements can be mapped to the same position, which potentially might make it impossible to assign the same valuation to two different packs of bits.

Here, we present a new and simple way of circumventing this obstacle by guessing the sizes of preimages in a mapping corresponding to a satisfying assignment. Less formally, what we do is replicate some positions and remove other ones, so that in some branch our guess will transform a mapping we had in mind into a permutation.

We would like to note that encoding groups of bits by a position in a permutation was already used by Marx, Lokshtanov and Saurabh [15] in the $k \times k$-Permutation Clique problem, as well as by Socała [17] in the lower bound for the Channel Assignment problem. Both of these two reductions (especially Lemma 2.3 from [17]) could be simplified when using our guessing preimage sizes approach, instead of a technical one-to-one reduction.

In the remainder of this section we prove Lemma 3.2, that is show how to transform a 3-SAT formula φ into $2^{\mathcal{O}(n/\log n)}$ instances of $(\mathcal{O}(1), \mathcal{O}(\log n))$-Subgraph Isomorphism with $\mathcal{O}(n/\log n)$ vertices. In order to do this we need to introduce notation for binary strings. Assume for a moment, that n is a power of two, i.e., $n = 2^k$ for $k \in \mathbb{N}$. One can view elements in a permutation as integers between 0 and $n-1$, denoted as $[n]$, but also as a set of binary strings of length k – being the binary representations of numbers from $[n]$, denoted as $2^{[k]}$. We will use the two conventions interchangeably and for this reason we need the following notation regarding binary strings. Let $\mathcal{B} := \{0,1\}^*$ be the set of all binary strings. and $\mathcal{B}_k := \{0,1\}^k$ be the set of binary strings of size exactly k. For a binary string s, let $|s|$ be its length. We denote the i-th digit (starting from 0) of a binary string s as s_i.

Proof of Lemma 3.2: Assume we are given a formula φ with n variables, such that each clause involves exactly three variables and each variable appears in at most 4 clauses. Define $k := \lceil \log n - \log\log n \rceil + 9$. We prove that solving φ can be reduced to solving less than $2^{2^{k+1}} = 2^{\mathcal{O}(n/\log n)}$ instances of $(3,k)$-Subgraph Isomorphism, with vertex colored denoted as red, green and blue, and edge colors denoted by $[k]$, where the number of vertices of both the pattern and host graph equals

$$2^k + 8 \cdot \binom{k}{3} + 1 = \mathcal{O}(n/\log n)\,.$$

Satisfying assignment gadget.
The assignment gadget G consists of a path on $8 \cdot \binom{k}{3}$ red vertices, with a single green vertex appended at one end. The red vertices will be uniquely identifiable based on the distance from the green vertex. Each red vertex will correspond to a choice of 3 distinct indices from $[k]$ and an assignment of binary values to each of them:

$$\begin{gathered}(i_1, i_2, i_3, b_1, b_2, b_3) \in [k]^3 \times \mathcal{B}_3\,,\\ i_1 < i_2 < i_3\,.\end{gathered}$$

Intuitively, an edge between one of the clause vertices and a red vertex will indicate that 'in this pack of clause valuations, the variables at positions i_1, i_2, i_3 are *not* assigned values b_1, b_2, b_3 at the same time'. All edges in G are of color 0.

Pattern graph construction.
The pattern graph will be constant across all the created instances. The pattern graph P consists of 2^k blue vertices corresponding to packs of clauses and a copy of the satisfying assignment gadget

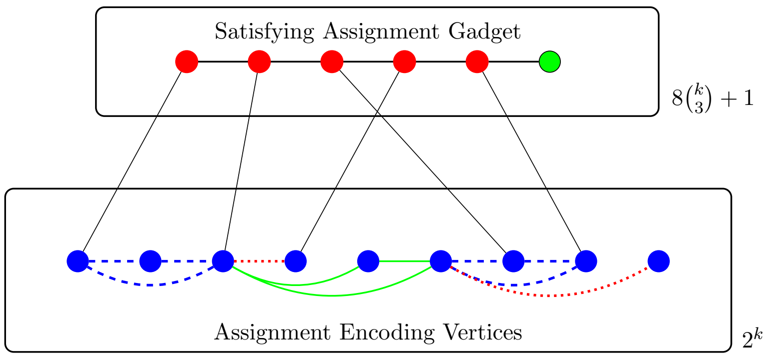

Figure 1: A simplified view of the pattern graph.

G. First, find the coloring l and packing h guaranteed by Lemma 4.2. We associate each blue vertex of H with a different group in the image of h.

For every variable x in S and every two distinct clauses C_1, C_2 containing x, we add an edge of color $l(x)$ between the blue vertices corresponding to $h(C_1)$ and $h(C_2)$. Intuitively, these edges signify that x has to have a consistent valuation when choosing valuations of variables in packs containing C_1 and C_2.

Additionally, for every clause C in φ we add an edge of color 0 between $h(C)$ (i.e., the pack containing C) and the red vertex $(i_1, i_2, i_3, b_1, b_2, b_3)$, where $i_1 < i_2 < i_3$ are the colors of variables in C and (b_1, b_2, b_3) is their only valuation that does not satisfy C.

Host graph construction.

We will generate a different host graph for every sequence of preimage sizes of the valuations of the groups. Fix a sequence $s_0, s_1, \ldots, s_{2^k-1}$, such that $s_i \geq 0$ for all i and $\sum s_i = 2^k$.

The number of possible such sequences s is

$$\binom{2^{k+1}-1}{2^k-1} \leq 2^{2^{k+1}}.$$

The host graph H_s consists of 2^k blue vertices corresponding to valuations of the groups of clauses and a copy of the satisfying assignment gadget G. For the binary string of length k corresponding to $i \in [2^k]$, we generate s_i vertices corresponding to it.

For $j \in \mathbb{Z}_k$, we join two blue vertices u, v in H with an edge of color j iff $u_j = v_j$, that is iff the j-th bit in both strings is the same. Intuitively, lack of an edge of color j between two blue vertices u, v in H disallows assigning two packs of clauses to vertices u and v when the variable of color j in both packs is the same, as it would lead to inconsistent valuation.

For every blue vertex u in H and red vertex $v = (i_1, i_2, i_3, b_1, b_2, b_3)$ in G, we connect u and v with an edge of color 0 iff $u_{i_j} \neq b_j$ for some $j \in \mathbb{Z}_3$. Less formally, lack of an edge between a blue vertex u and a red vertex $v = (i_1, i_2, i_3, b_1, b_2, b_3)$ means that a pack of clauses can be assigned to u, only if the valuation corresponding to the bit string associated with u only if there is no clause such that assigning values b_1, b_2, b_3 to variables of colors i_1, i_2, i_3, respectively, would cause come clause from the pack to be unsatisfied.

Proof of correctness 3-SAT.

As the construction can be carried out in polynomial time per instance and both the host and pattern graphs have $\mathcal{O}(n/\log n)$ vertices as promised, it remains to prove that φ is satisfiable iff for some instance the pattern graph P is a subgraph of the host graph H_s.

Claim 5.1 *If φ is satisfiable, then for some sequence of preimage sizes s, P is a subgraph of H_s.*

Proof: First, assume that φ is satisfiable and let $\mathrm{val} : \mathrm{Var}(\varphi) \to \{\mathbf{true}, \mathbf{false}\}$ be a satisfying assignment. We construct a mapping $g : V(P) \to V(H)$ as follows. For a group $i \in [2^k]$ let Var_i be the set of variables occurring in all the clauses assigned to i by the packing h. If any colors do not occur in $l(\mathrm{Var}_i)$, add arbitrary variables to Var_i so that $l(\mathrm{Var}_i) = [k]$. Define $f(i) = \mathrm{val}|_{\mathrm{Var}_i}$, i.e., the bit string representing valuation of variables from Var_i by val. Let s be the sequence of preimage sizes of f. A bijection g corresponding to f exists between the blue vertices of P and H_s. We extend g to all the vertices of P by mapping each vertex of the satisfying assignment gadget G in P to its corresponding copy in H_s, obtaining a bijection $g' : V(P) \to V(H)$.

It remains to check that b' preserves all the edges. Clearly, the edges within the satisfying assignment gadget G are preserved. Consider any edge of color $c \in [k]$ in the pattern graph between two blue vertices u, v, corresponding to groups i and j. By construction, this means that the packs $h^{-1}(i)$ and $h^{-1}(j)$ share a variable of color c, which means that by the definition of f the bit strings $f(i)$ and $f(j)$ assign the same value to the index corresponding to this variable. As g extends f, we have $g(i) = f(i)$ and $g(j) = f(j)$, hence the bit strings corresponding to $g'(u)$ and $g'(v)$ have the same value on the c-th position, hence by construction of the host graph $g'(u)$ and $g'(v)$ are connected by an edge of color c. Finally, we inspect the edges between blue vertices and red vertices. Consider a blue vertex u associated with a set $A \subseteq [k]$, which is connected to some red vertex $v = (i_1, i_2, i_3, b_1, b_2, b_3)$, because of a clause $C \in h^{-1}(A)$. As val is a satisfying assignment and g extends bit strings assigned by f, we infer that the vertex $g(u)$ is connected to the red vertex v. Consequently, P is a subgraph of H_s witnessed by the mapping g'. ⌟

In Appendix B we prove the following claim.

Claim 5.2 (♠) *If for any s it holds that P is a subgraph of H_s, then φ is satisfiable.*

Claims 5.1 and 5.2 prove equivalence of the formula φ and created instances of $(3, k)$-Subgraph Isomorphism, hence the proof of Lemma 3.2 follows. ■

We would like to note that Lemma 3.2 implies a tight bound for the auxiliary version of Subgraph Isomorphism with colors, even in the case when the number of vertex colors is constant and the number of edge colors is logarithmic.

Corollary 5.3 *There is no $2^{o(n \log n)}$ time algorithm for the $(\mathcal{O}(1), \mathcal{O}(\log n))$-Subgraph Isomorphism problem, unless the Exponential Time Hypothesis fails.*

6 Removing the colors

In this section we prove Lemma 3.3, first by showing how to remove colors from edges, and next by removing colors from vertices. Due to space constraints, we only sketch the constructions, and the formal proof of the equivalence of created instances is deferred to Appendix C.

Lemma 6.1 (♠) *An instance (G, H) of (c, t)-Subgraph Isomorphism such that $|V(G)| = |V(H)|$ can be reduced to an instance (G', H') of $(c+1, 1)$-Subgraph Isomorphism with $\mathcal{O}(\sqrt{t})$ times more vertices such that $|V(G')| = |V(H')|$.*

Sketch of proof: Let (G, H) be an instance of (c, t)-Subgraph Isomorphism such that $|V(G)| = |V(H)|$. Assume that none of the vertices of the instance (G, H) was colored yellow. Let $t' := 2\lceil\sqrt{t}\rceil$. Note that for $t \geq 1$ we have $\lceil\sqrt{t}\rceil \geq 1$ and then

$$\binom{t'}{2} = \frac{2\lceil\sqrt{t}\rceil \cdot (2\lceil\sqrt{t}\rceil - 1)}{2} \geq \lceil\sqrt{t}\rceil^2 \geq t.$$

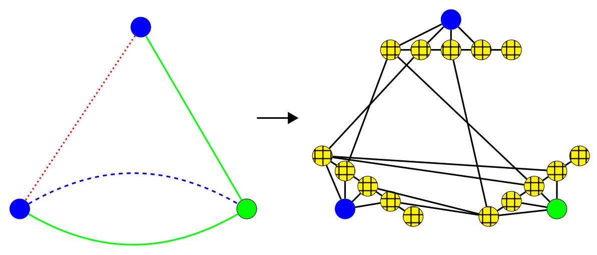

Figure 2: The reduction described in Lemma 6.1. In the example, $t = 3$, $t' = 2\lceil\sqrt{t}\rceil = 4$, $p(\text{red}) = (1, 2)$, $p(\text{green}) = (1, 3)$ and $p(\text{blue}) = (1, 4)$.

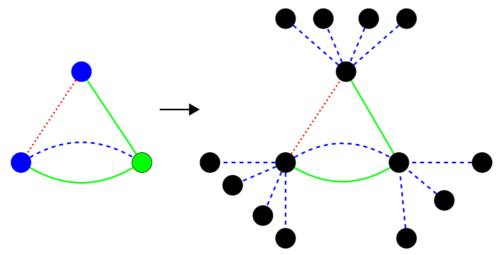

Figure 3: The reduction described in Lemma 6.2. In the example, the green and blue colors of the vertices represent the numbers 2 and 3 respectively and the blue color of the edges represent the number 1.

Therefore for each color $x \in [t]$ we can pick a different pair $p(x) := (i, j)$ where $1 \leq i < j \leq t'$.

For every vertex u in either the pattern or the host graph, we replace it by a gadget consisting of $t' + 2$ vertices (see Fig. 2):

- a *center vertex* u'_0 of the same color as u, and
- a path on $t' + 1$ yellow vertices $u'_1, \ldots, u'_{t'+1}$, the first t' of which are connected to the center vertex.

For every edge (u, v) of color x in either the pattern or the host graph, we replace it by the edges (u'_i, v'_j) and (u'_j, v'_i) in the modified graph, where $(i, j) = p(x)$. We denote this new instance of $(c + 1, 1)$-Subgraph Isomorphism as (G', H'). Note that $|V(G')| = (t' + 2) \cdot |V(G)|$ and $|V(H')| = (t' + 2) \cdot |V(H)|$ hence $|V(G')| = |V(H')|$ and also $|V(G')| = \mathcal{O}(\sqrt{t}) \cdot |V(G)|$ and $|V(H')| = \mathcal{O}(\sqrt{t}) \cdot |V(H)|$. In Appendix C we show that G is a subgraph of H iff G' is a subgraph of H'. ■

Having reduced the number of edge colors down to one, it remains to reduce the number of vertex colors. Note that in the following lemma it would be enough to assume $t = 1$, however we prove the lemma in a more general form as it does not affect the complexity of the proof.

Lemma 6.2 (♠) *An instance (G, H) of (c, t)-Subgraph Isomorphism such that $|V(G)| = |V(H)|$ can be reduced to an instance (G', H') of $(1, t)$-Subgraph Isomorphism with $\mathcal{O}(c)$ times more vertices such that $|V(G')| = |V(H')|$.*

Sketch of proof: Let (G, H) be an instance of (c, t)-Subgraph Isomorphism such that $|V(G)| = |V(H)|$. Number the vertex colors arbitrarily from 1 to c and number the edge colors arbitrarily from 1 to t. We can assume that for every vertex color the number of the vertices in this color in G and in H is the same because otherwise we can produce a trivial NO instance as (G', H'). In both pattern and host graphs, for each vertex v, attach $i+1$ new leaves $v_1, v_2, \ldots, v_{i+1}$ to it, where i is the color of v, using edges of color 1 (or any fixed color from 1 to t). We also denote $v_0 = v$. Consider the $(1,t)$-Subgraph Isomorphism instance (G', H') on the new graphs. For every vertex color the number of the vertices in that color in G is the same as in H and therefore the number of added leaves is the same in G' as in H'. Hence $|V(G')| = |V(H')|$. In Appendix C we show G is a subgraph of H iff G' is a subgraph of H'. ■

Proof of Lemma 3.3: The thesis follows directly from consecutive application of Lemmas 6.1 and 6.2. ■

7 From Graph Homomorphism to Subgraph Isomorphism

Graph Homomorphism
Input: undirected graphs G, H.
Question: Is there a homomorphism from G to H, i.e., does there exist a function $h : V(G) \to V(H)$, such that for each edge $uv \in E(G)$ we have $h(u)h(v) \in E(H)$.

In this section we present a reduction which shows that one can solve the Graph Homomorphism problem by solving $2^{|V(G)|+|V(H)|}$ instances of the Subgraph Isomorphism problem, demonstrating that the lower bound of $2^{\Omega(|V(G)| \log V(H)/\log\log V(H))}$ of Fomin et al. [9] implies an $2^{\Omega(n \log n/\log\log n)}$ lower bound under the Exponential Time Hypothesis for the Subgraph Isomorphism problem, where $n = |V(G)| + |V(H)|$.

Proof of Theorem 1.3: Let (G, H) be an instance of Graph Homomorphism and denote $n = V(G) + V(H)$. Note that any homomorphism h from G to H can be associated with some sequence of non-negative numbers $(|h^{-1}(v)|)_{v \in V(H)}$, being the numbers of vertices of G mapped to particular vertices of H. The sum of the numbers in such a sequence equals exactly $|V(G)|$. As the number of such sequences is $\binom{V(G)+V(H)-1}{V(H)-1} \leq 2^n$, we can enumerate all such sequences in time $2^n \mathrm{poly}(n)$. For each such sequence $(a_v)_{v \in V(H)}$ we create a new instance (G', H') of Subgraph Isomorphism, where the pattern graph remains the same, i.e., $G' = G$, and in the host graph H' each vertex of $v \in V(H)$ is replicated exactly a_v times (possibly zero). Observe that $|V(H')| = |V(G')|$.

We claim that G admits a homomorphism to H iff for some sequence $(a_v)_{v \in V(H)}$ the graph G' is a subgraph of H'. First, assume that G admits a homomorphism h to H. Consider the instance (G', H') created for the sequence $a_v = |h^{-1}(v)|$ and observe that we can create a bijection $h' : V(G') \to V(H')$ by assigning $v \in V(G')$ to its private copy of $h(v)$. As h is a homomorphism, so is h', and as h' is at the same time a bijection, we infer that G' is a subgraph of H'.

On the other hand if for some sequence $(a_v)_{v \in V(H)}$ the constructed graph G' is a subgraph of H', then projecting the witnessing injection $g : V(G') \to V(H')$ so that $g'(v)$ is defined as the prototype of the copy $g(v)$ gives a homomorphism from G to H, as copies of each $v \in V(H)$ form independent sets in H'. ■

References

[1] School on parameterized algorithms and complexity - open problems. In *http://fptschool.mimuw.edu.pl/opl.pdf*, page 8, 2014. 1

[2] O. Amini, F. V. Fomin, and S. Saurabh. Counting subgraphs via homomorphisms. *SIAM J. Discrete Math.*, 26(2):695–717, 2012. 1, 1

[3] R. Beigel and D. Eppstein. 3-coloring in time $o(1.3289^n)$. *J. Algorithms*, 54(2):168–204, 2005. 1

[4] A. Björklund. Determinant sums for undirected Hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. 1

[5] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. 1

[6] N. Bourgeois, B. Escoffier, V. T. Paschos, and J. M. M. van Rooij. Fast algorithms for max independent set. *Algorithmica*, 62(1-2):382–415, 2012. 1

[7] M. Cygan and M. Pilipczuk. Bandwidth and distortion revisited. *Discrete Applied Mathematics*, 160(4-5):494–504, 2012. 1

[8] F. Fomin, K. Iwama, and D. Kratsch. Moderately Exponential Time Algorithms (Dagstuhl Seminar 08431). In *Dagstuhl Reports, http://drops.dagstuhl.de/opus/volltexte/2008/1798/pdf/08431.SWM.Paper.1798.pdf*, page 1, 2008. 1

[9] F. V. Fomin, A. Golovnev, A. S. Kulikov, and I. Mihajlin. Lower bounds for the graph homomorphism problem. *CoRR*, abs/1502.05447, 2015. 1, 1, 1.2, 7

[10] F. V. Fomin and D. Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010. 1

[11] T. Husfeldt, R. Paturi, G. B. Sorkin, and R. Williams. Exponential Algorithms: Algorithms and Complexity Beyond Polynomial Time (Dagstuhl Seminar 13331). In *Dagstuhl Reports, http://drops.dagstuhl.de/opus/volltexte/2013/4342/pdf/dagrep_v003_i008_p040_s13331.pd*
page 63, 2013. 1

[12] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. 1, 2, 2.3

[13] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. 1, 2, 2.2

[14] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. 1

[15] D. Lokshtanov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 760–776, 2011. 5

[16] D. Marx and M. Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 542–553, 2014. 1

[17] A. Socała. Tight lower bound for the channel assignment problem. In *SODA*, 2015. 5

[18] C. A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85 – 89, 1984. 2.1

A Missing proofs from Section 4

Before we prove Lemma 4.1, we show the existence of a potentially unbalanced 9-coloring.

Lemma A.1 *Given a* 3−SAT *formula φ with n variables, where each variable occurs in at most 4 clauses, we can color the variables of φ in polynomial time using at most 9 colors, so that no clause contains two variables of the same color.*

Proof: Construct an auxiliary graph G_φ, the vertex set of which is the set of variables of S, where two vertices of G_φ are adjacent iff they both appear in at least one of the clauses of φ. Note that the maximum degree of G_φ is bounded by 8, as each variable appears in at most 4 clauses and each clause contains at most 3 literals. Consequently, we can color G_φ with at most 9 colors in a greedy manner. ■

Proof of Lemma 4.1: First, color the variables into 9 colors using Lemma A.1. Then, while there exists a color with more than $\lceil n/(k-9) \rceil$ variables assigned to it, separate $\lceil n/(k-9) \rceil$ of them to form a new color. This can occur at most $k-9$ times, and the lemma follows. ■

B Missing proofs from Section 5

Proof of Claim 5.2: Let g be a mapping from P to H_s witnessing the fact that P is a subgraph of H_s. As g respects colors, we infer that the single green vertex in P is mapped to the single green vertex in H_s. Similarly all the red vertices of P have to be mapped to red vertices of H_s. Additionally the distance between each red vertex v and the green vertex in P cannot be smaller than the distance between $g(u)$ and the green vertex in H_s. As red vertices induce a path, and the green vertex is pendant to one if its ends, we infer that g assigns each vertex of the satisfying-assignment-gadget in P to its copy in H_s (in short, by construction there are no non-trivial automorphisms of the gadget).

Construct an assignment val : $\mathrm{Var}(\varphi) \to \{\textbf{true}, \textbf{false}\}$ as follows. For a variable $x \in \mathrm{Var}(\varphi)$ find any clause C that contains x and assign val(x) to **true** iff $g(h(C))_{l(x)} = 1$, where $h(C)$ is the blue vertex associated with C and $l(x)$ is the color of the variable x. Note that by construction the assignment val is well-defined, as edges between blue vertices guarantee consistence. Consider a clause C. The edges between $h(C)$ and red vertices in the pattern graph P have to be preserved by g, and we already observed that g maps red vertices of P to their corresponding copies in H_s. Hence, we infer that there is an edge between $g(h(C))$ and the red vertex $v = (i_1, i_2, i_3, b_1, b_2, b_3)$, where b_1, b_2, b_3 is the only assignment to variables of C, where $l(C) = \{i_1, i_2, i_3\}$, which does not satisfy C. This in turn implies that for at least one variable of C the assignment val assigns a different value than the one corresponding to the appropriate bit from $\{b_1, b_2, b_3\}$. Consequently, val is a satisfying assignment. ⌟

C Missing proofs from Section 6

Here, we present the missing parts of the proof of Lemmas 6.1 and 6.2 from Section 6.

Proof of Lemma 6.1: It remains to prove that G is a subgraph of H iff G' is a subgraph of H'. If G is a subgraph of H then there exists an injective function $f : V(G) \to V(H)$ such that edges and colors are preserved. Note that every vertex of the instance (G', H') is of the form u'_i for some vertex u of the instance (G, H). Let $g : V(G') \to V(H')$ be a function such that $g(u'_i) = f(u)'_i$. The function g is an injection because the function f is an injection. The function g preserves the colors of the vertices because $col(g(u'_0)) = col(f(u)'_0) = col(f(u)) = col(u) = col(u'_0)$

and for $i > 0$ the color of u'_i is always yellow. The function g preserves also the edges. Let $u'_i v'_j$ be an edge in G' such that $i \leq j$ (we can assume this w.l.o.g.). If $u = v$ then there exists also an edge $f(u)'_i f(u)'_j = g(u'_i)g(v'_j)$ in H' because all the gadgets have exactly the same structure of the internal edges i.e. if there exists an edge $u'_i u'_j$ in a gadget for any vertex u then for every vertex v there exists an edge $v'_i v'_j$ in a gadget for vertex v. If $u \neq v$ then $1 \leq i < j \leq t'$ and there exists an edge uv of the color $x = p^{-1}(i, j)$ in G and then there exists an edge $f(u)f(v)$ of the color x in H and (because $(i, j) = p(x)$) we know that there exists an edge $f(u)'_i f(v)'_j = g(u'_i)g(v'_j)$ in H'. The edges of (G', H') have only one color thus g preserves the colors of the edges trivially. Hence G' is a subgraph of H'.

If G' is a subgraph of H' then there exists an injective function $g : V(G') \to V(H')$ such that edges and colors are preserved. The vertices of the form u'_0 are the only vertices of G' and H' which are not yellow. Therefore if $g(u'_i) = v'_j$, then $i = 0$ iff $j = 0$. If $g(u'_0) = v'_0$ then for every u'_i such that $1 \leq i \leq t'$ we have $g(u'_i) = v'_j$ for some $1 \leq j \leq t'$ because the vertices $u'_1, u'_2, \ldots, u'_{t'}$ are yellow neighbors of u'_0 and the vertices $v'_1, v'_2, \ldots, v'_{t'}$ are the only yellow neighbors of v'_0. On the other hand we know that $|V(G)| = |V(H)|$ and then the number of the vertices of the form u'_i for $1 \leq i \leq t'$ is the same in G' and in H'. Therefore for every vertex v'_i in H' such that $1 \leq i \leq t'$ there exists a vertex u'_j in G' such that $1 \leq j \leq t'$ and $g(u'_j) = v'_i$. Therefore if $g(u'_i) = v'_j$ then $i = t'+1$ iff $j = t'+1$. Moreover the vertices $u'_1, u'_2, \ldots, u'_{t'}$ create a path (in this order) and the only directed paths containing exactly the vertices $\{v'_1, v'_2, \ldots, v'_{t'}\} = g(\{u'_1, u'_2, \ldots, u'_{t'}\})$ are $v'_1, v'_2, \ldots, v'_{t'}$ and $v'_{t'}, v'_{t'-1}, \ldots, v'_1$. But the vertex $u'_{t'}$ is a neighbor of the vertex $u'_{t'+1}$ and the vertex v'_1 has no neighbor of the form $w'_{t'+1}$ for any vertex w in H. On the other hand the vertex $u'_{t'+1}$ has to be mapped to a vertex of the form $w'_{t'+1}$ for some vertex w in H. Therefore the path $u'_1, u'_2, \ldots, u'_{t'}$ is mapped to the path $v'_1, v'_2, \ldots, v'_{t'}$ i.e. for every vertex u'_i such that $1 \leq i \leq t'$ we have $g(u'_i) = v'_i$. Let $f : V(G) \to V(H)$ be a function such that $f(u) = v$ iff $g(u'_0) = v'_0$. (note that then $f(u)'_0 = v'_0 = g(u'_0)$). The function f is an injection because the function g is an injection. The function f preserves the colors of the vertices because $col(f(u)) = col(f(u)'_0) = col(g(u'_0)) = col(u'_0) = col(u)$. The function f preserves also the edges with their colors because if there is an edge uv of the color x in the graph G then for $(i, j) = p(x)$ there is an edge $u'_i v'_j$ in the graph G' and therefore there is an edge $g(u'_i)g(v'_j) = f(u)'_i f(v)'_j$ in the graph H' and then there is an edge $f(u)f(v)$ of the color x in the graph H. Hence G is a subgraph of H. ■

Proof of Lemma 6.2: If G is a subgraph of H then there exists an injective function $f : V(G) \to V(H)$ such that edges and colors are preserved. Note that every vertex of the instance (G', H') is of the form v_i for some vertex v of the instance (G, H). Let $g : V(G') \to V(H')$ be a function such that $g(v_i) = f(v)_i$ which is a correctly defined function because $col(v) = col(f(v))$ and therefore v_0 has the same number of leaves in the graph G' as $f(v)_0$ in the graph H'. The function g is an injection because the function f is an injection. The function g preserves the colors of the vertices trivially. We show that the function g preserves also edges and their colors. Let assume that there is an edge $u_i v_j$ for $i \leq j$ (we can assume that w.l.o.g) of the color x in the graph G'. If $j > 0$ then $u = v$, $i = 0$ and $x = 1$ and there exists also an edge $f(u)_0 f(u)_j = g(u_i)g(v_j)$ of the color $1 = x$ in the graph H'. Otherwise we have $i = j = 0$ and then there exists an edge uv of the color x in the graph G thus there exists an edge $f(u)f(v)$ of the color x in the graph H hence there exists an edge $f(u)_0 f(v)_0 = g(u_i)g(v_j)$ of the color x in the graph H'. Therefore G' is a subgraph of G.

If G' is a subgraph of H' then there exists an injective function $g : V(G') \to V(H')$ such that edges and colors are preserved. All vertices from the original pattern graph have to be matched to vertices from the original host graph, as they are the only ones of degree greater than 1 in the new graphs. But the number of the vertices of the form u_0 is the same in G' as in H' because $|V(G)| = |V(H)|$. Therefore for every vertex of the form v_0 in H' there exists a vertex of the form u_0 in G' such that $g(u_0) = v_0$. Hence, the leaves have to map to leaves. But the number of leaves is the same in G' as in H'. Thus for every leaf v_i in H' there exists a leaf u_j in G' such that $g(u_j) = v_i$. Hence all the leaves are used and then for every vertex u_0 in G' the number of leaves of u_0 in G' is the same as the number of leaves of $g(u_0)$ in H'. Let us consider a function $f : V(G) \to V(H)$ such that $f(u) = v$ iff $g(u_0) = v_0$ (then $f(u)_0 = v_0 = g(u_0)$). Note that $f = g|_{V(G)}$. The function f is an injection because the function g is an injection. The function f preserves the colors of the vertices because for every v in the graph G we have that v_0 has exactly $col(v)+1$ leaves as neighbors in the graph G' and then $g(v_0) = f(v)_0$ has also exactly $col(v) + 1$ leafs as neighbors. But on the other hand the vertex $f(v)_0$ has exactly $col(f(v)) + 1$ leafs as neighbors in the graph H' and therefore $col(f(v)) = col(v)$. The function f preserves also the edges with their colors because for every edge uv of a color x in the graph G there exists an edge u_0v_0 of the color x in the graph G' and therefore there exists an edge $g(u_0)g(v_0) = f(u)_0f(v)_0$ of the color x in the graph H' and hence there exists an edge $f(u)f(v)$ of the color x in the graph H. Therefore G is a subgraph of H. ■